Introduction to Genetic Algorithms

Max Peysakhov
About Genetic Algorithms

• In the 1950’s and 1960’s computer scientists started to study the evolutionary optimization systems. Genetic Algorithms were one of them.

• Genetic Algorithms (GA) operate on the set (population) of candidate solutions (individuals are also called chromosomes).

• Chromosomes are the strings or arrays of genes (a gene is the smallest building block of the solution).

• Each iteration of the search is called a generation.

• The idea was to evolve a solution of high quality from the population of candidate solutions.
Architecture of the Genetic Algorithm

- **Initialization**: Initial Population
- **Evaluate Chromosomes**: Evaluated Population
- **Select Chromosomes**: Candidate Next Generation
- **Crossover Chromosomes**: Candidate Next Generation
- **Mutate Chromosomes**: Next Generation
- **Evaluate Chromosomes**: Best solution found so far

Data flow: Data → Initialization → Evaluate Chromosomes → Select Chromosomes → Crossover Chromosomes → Mutate Chromosomes → Evaluate Chromosomes → Best solution found so far
Illustration of the Genetic Algorithm

- Evaluation Function
- Initial Population
- Final Population
- Individual
- Best individual solution
• You begin with a population of random bit strings.
• Each bit string encodes some problem configuration.
• Example: You can encode SAT problem by representing each Boolean variable by a position in the bit string

\[(a_0 \lor \neg a_1 \lor a_2) \land (\neg a_0 \lor a_2 \lor a_3) \land (\neg a_0 \lor a_2 \lor \neg a_3) \land (\neg a_2 \lor \neg a_3 \lor \neg a_4) \land (\neg a_0 \lor a_2 \lor a_4)\]

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 \\
\ldots \\
1 & 0 & 1 & 0 \\
\end{array}
\]

\(N\) randomly generated individuals (initial population)
**Some Terminology**

Genotype – Configuration of the bits in the string.

![Genotype example]

Phenotype – What genotype represents.

\[a_0 = T, \ a_1 = F, \ a_2 = T, \ a_3 = F, \ a_4 = T\]

Fitness Landscape – graph of the fitness function
Fitness Function

• In Nature: **THE FITTEST SURVIVE**

• In GA we have to find out how to measure the fitness.

• Fitness function should guide algorithm to the optimal solution.

• Example: For SAT problem we can use the number of satisfied clauses as fitness function.
• Ones we evaluated the individuals how do we choose ones that survive?
• Each individual gets a chunk of a roulette wheel proportionate to it’s fitness relative to the fitness of others.
• Spin the roulette wheel \( N \) times.
Selection: Universal Stochastic Sampling

- Than using Fitness Proportional Selection it is possible (although unlikely) that all selected individuals will have lowest fitness.
- To fix that problem USS suggests to spin \(N\)-pointer roulette wheel only once.
Some individuals may be much more fit than others. Their offspring will dominate the population and can cause a convergence on a local maximum.

Sigma Scaling is designed to keep a selection’s pressure relatively constant over the run of GA.

\[
\text{Exp.Val} = \text{iif} (\text{Stddev} = 0, 1, 1+ (f(x) - \text{Mean}(f)) / 2*\text{Stddev})
\]
Selection: Rank proportional

- Same purpose as Sigma scaling.
- All individuals in the population are ranked in order of their fitness.
- By adjusting Max desirable constant selection pressure can be achieved
- \( \text{Min} + \text{Max} = 2 \) and \( 1 < \text{Max} < 2 \)

\[
\text{Exp. Val} = \text{Min} + (\text{Max}-\text{Min}) \times ((\text{Rank } (I) - 1) / (N-1)).
\]
Crossover: Single Point Crossover

- Two parents selected at random.
- Single crossover point selected at random.
- Chromosomes are cut at the crossover point.
- Tail part of each chromosome spliced with head part of the other chromosome.

Parent 1: 11110
Parent 2: 00101
Offspring 1: 11101
Offspring 2: 00110
Crossover: K-Point Crossover

- Two parents selected at random.
- $k$ crossover points selected at random.
- Chromosomes are cut at the crossover points.
- Appropriate sections are swapped.

Parent 1

```
1 1 1 1 0
```

Crossover points

```
0 0 1 0 1
```

Parent 2

```
1 1 1 1 0
```

Offspring 1

```
1 0 1 1 1
```

Offspring 2

```
0 1 1 0 0
```
Crossover: Uniform Crossover

- Two parents selected at random.
- Iterate through chromosomes.
- With probability $p_c$ swap individual genes.

<table>
<thead>
<tr>
<th>Parent 1</th>
<th>Offspring 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 0</td>
<td>0 1 1 1 1</td>
</tr>
<tr>
<td>↓  ↓  ↓  ↓</td>
<td>↓  ↓  ↓  ↓</td>
</tr>
<tr>
<td>0 0 1 0 1</td>
<td>1 0 1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parent 2</th>
<th>Offspring 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 0</td>
<td>0 1 1 1 1</td>
</tr>
<tr>
<td>↓  ↓  ↓  ↓</td>
<td>↓  ↓  ↓  ↓</td>
</tr>
<tr>
<td>0 0 1 0 1</td>
<td>1 0 1 0 0</td>
</tr>
</tbody>
</table>
- Iterate through chromosomes.
- With probability $p_c$ alter individual genes.

```
1 1 1 1 0
```

Mutated gene

```
1 1 0 1 0
```
Elitism

- Trick that works.
- Select small number of the most fit individuals.
- Copy them to the new generation unchanged.
- Preserve valuable members of the population
When to Stop?

- When you found an optimal solution.
- When you completed the predefined number of generations.
- When time limit expired.
- When population converges to a single individual.
- When fitness function converges to a single value.
- After some number of generations with no improvement.
- ... Any other ideas?
- Usually use combination of 2 or 3 stopping criteria.
1. Let $P$ = a random population of $N$ individuals.

2. While Stopping criteria is false do:

3. Let $f_i = \text{Fitness}(P_i)$ for $\forall P_i \in P$

4. Let $P' = \text{Select\_New\_Population}(P,f)$

5. Let $P' = \text{Crossover}(P')$

6. Let $P' = \text{Mutate}(P')$

7. Let $P = P'$

---

**Simple GA**
Back to Our Example

Let’s simulate one iteration of GA with population size 4

\[(a_0 \lor \neg a_1 \lor a_2) \land (\neg a_0 \lor a_2 \lor a_3) \land (\neg a_0 \lor a_2 \lor \neg a_3) \land (\neg a_2 \lor \neg a_3 \lor \neg a_4) \land (\neg a_0 \lor a_2 \lor a_4)\]
Parameter Tuning

- We defined large number of control parameters.
  - Crossover rate
  - Mutation rate
  - Stopping Criteria
  - Population Size.
  - Any others?

- How do we set these?

- ... Any ideas?

- Wide open research area.

- No general set of rules or guidelines.

- Most often used method is trial-and-error.
Parameter Tuning (2)

- Meta-level optimization.
  - Employing a second GA to optimize parameters of the first one.
  - Fitness evaluation is expensive.
  - Difficult to identify fitness.
- Adapting control parameters.
  - Adapting control parameters over time.
  - Using problem related feedback to adjust parameters.
“Messy” Genetic Algorithms

- Length of the chromosome may change during the execution.
- Achieved by using messy crossover operators.
  - Cut - cuts chromosome in arbitrary point.
  - Splice - joins two parts of the chromosome together.
- Used then length of the solution is unknown.

Parent 1

```
1 1 1 1 1 0
```

Offspring 1

```
1 1 1 1 0 1
```

Cut points

```
0 0 1 0 1
```

Parent 2

```
0 0 1 0 1
```

Offspring 2

```
0 0 1 0
```
Genetic Programming (GP)

- Formulated by Koza in 1990 as means of automatic programming
- Similar to GA but:
  - Chromosomes are encoded as parse trees typically representing a computer program in Lisp

```
(if (> x y) x y)
```

Diagram:
```
    IF
   /   
  >    
 /     
X     Y
```

X
```
   / 
  X   Y
```
GP: Initialization

• Define your function set $C$
  • $\{+, -, *, /, \text{ABS, if}, \ldots \}$
• Define your terminal states $T$
  • $\{0, 1, 2, \ldots, x, y, z, \ldots, \pi, e, \ldots \}$
• For each population member
  1. Start with empty parse tree $P$
  2. Add $X_i$ to $P$ such that $X_i \in C$ or $X_i \in T$
  3. Repeat procedure recursively for all children of $X_i$
GP: Crossover

- Two parents selected at random.
- Random nodes independently picked in each parent.
- Swap sub-trees originated at these nodes.

\[
\text{IF } (x > y) \Rightarrow x + y
\]

Crossover points

\[
\text{IF } (x > y) \Rightarrow x y
\]

(* (+ x y) y)

(* y y)
GP: Mutation

- One chromosome selected at random.
- Random node picked.
- Remove sub-tree rooted at that node.
- Grow a random sub-tree out of this node

\[
\text{(if (> \text{x} \text{y}) \text{x} \text{y})}
\]

 Mutated gene

\[
\text{(if (> \text{x} \text{y}) (- \text{x} \text{y}) \text{y})}
\]
• Algorithms capable of trading off execution time for solution quality called anytime algorithms.

• GA are naturally anytime algorithms.
Knapsack Problem

- **Definition:** Given items of different values and weights, find the most valuable set of items which fit in a knapsack of fixed weight limit.
- Classic NP hard, combinatorial optimization problem with a wide range of applications.
- Conventional search techniques may take too long to produce a result of any value.
- Let’s set up a GA to solve this problem.
Knapsack Problem: Choosing an Algorithm.

- **Choices:**
  - Genetic Algorithm
  - Messy Genetic Algorithm
  - Genetic Programming
Knapsack Problem: Choosing a Representation.

- **Answer:** Messy Genetic algorithms, because the solution length is unknown up front.

- Given a set (array) of items $S_0$ to $S_N$ Find a suitable representation of the chromosome

<table>
<thead>
<tr>
<th></th>
<th>$S_0$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
<th>$S_8$</th>
<th>$S_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(s)$</td>
<td>64</td>
<td>59</td>
<td>35</td>
<td>27</td>
<td>31</td>
<td>34</td>
<td>43</td>
<td>63</td>
<td>62</td>
<td>15</td>
</tr>
<tr>
<td>$W(s)$</td>
<td>13</td>
<td>75</td>
<td>32</td>
<td>53</td>
<td>83</td>
<td>5</td>
<td>7</td>
<td>37</td>
<td>41</td>
<td>48</td>
</tr>
</tbody>
</table>
Knapsack Problem: Choosing Crossover Operators.

- **Answer:**
  
  **Genotype:** 0 1 5 6 2 8
  
  **Phenotype** - items $S_0, S_1, S_5, S_6, S_2, S_8$
  with total weight - 326
  with total value - 142

- Choose a crossover operators:
  - Single point
  - K-point, what’s the k
  - Uniform
Knapsack Problem: Choosing Mutation Operators.

• **Answer:** Got you! We are using messy GA, so we have to use messy crossover (cut and splice) operators.

• Create a mutation operator.
Knapsack Problem: Choosing Mutation Operators.

- **Answer:** that’s easy. Simply replace mutated gene with a random number from 1 to N

- Let’s see how it works:

  0 1 5 6 2 8
  \[\downarrow \quad \uparrow\text{Cut points}\]
  5 3 2 4 7 10

  \[\uparrow\text{Mutated gene}\]

  5 1 5 6 4 7 10

  \[\downarrow\]

  5 3 2 2 8

  - Can we fix it

  Oops!
Knapsack Problem: Choosing Evaluation Function.

- **Answer:** Sure. Delete all duplicate entries.

  | 5 | 1 | 5 | 6 | 4 | 7 | 10 |
  | 5 | 1 | 6 | 4 | 7 | 10 |

  | 5 | 3 | 2 | 2 | 8 |
  | 5 | 3 | 2 | 8 |

- Let’s think about evaluation function

  ?
Knapsack Problem: Choosing Evaluation Function. (2)

- **Answer:** The solution worth the sum of its items prices, if it is within weight limits, otherwise it worth 0

  \[ \text{if} \ (\text{Sum}(W(S_i)) > W_{kn}, 0, 1) \times \text{Sum}(P((S_i)) \]

- That means that all the solutions with access weight treated by GA in the same way.

- Shouldn’t we assign more utility to the lighter solutions than to the heavier ones?

- Can we improve the evaluation function.
Knapsack Problem: Choosing Selection Strategy

• **Answer:** The solution worth the sum of it’s items prices, if it is within weight limits, otherwise it’s fitness inverse proportionate to it’s weight.

  \[ \text{iif ( } \sum(W(S_i)) > W_{kn}, \frac{1}{\sum(W(S_i)), \sum(P(S_i))} \text{) } \]

• Choosing selection strategy
  • Fitness proportional
  • USS
  • Sigma Scaling
  • Rank
  • Should we perform Elitism
Answer: How do I know?
- Elitism usually improves performance.
- Although the Fitness Proportionate selection is often the worst, only experiments will show the best selection strategy.

Parameters like crossover and mutation rates, population size and others have to be tuned after the problem is implemented in code.

I would start with:
- Crossover rate 0.75
- Mutation rate 0.01
- Stopping Criteria 1000 Generations
- Population Size 100
GA Issues

- Choice of representation is critical.
- Choice of generic operators is critical.
- Design of fitness function is critical.
- Parameter tuning is critical.
- There is no template for doing it correctly from the first attempt.
GA Discussions

- Often a second best way to solve a problem.
- Easy to implement
- GA is uninformed search
- Can be improved by using heuristic operators
Evolving Virtual Creatures by Karl Sims

• Creating virtual creatures that move in simulated three-dimensional physical worlds.

• The morphologies of creatures and the neural systems for controlling their muscles are both generated using genetic algorithms.

• Different fitness evaluation functions are used to direct simulated evolutions towards specific behaviors such as swimming, walking, jumping, and following.

• More info at: http://www.genarts.com/karl/
- Simple paddling and tail wagging creatures.
- Some used symmetrical flippers to propel themselves.
- Water snake creatures evolved that wind through the water.
Evolving Virtual Walkers and Jumpers by Karl Sims

- Simple creatures that could shuffle or hobble along
- Some simply wag an appendage in the air to rock back and forth.
- More complex creatures push or pull themselves along.
- A number of simple jumping creatures did emerge.

Creatures evolved for walking.

Creatures evolved for jumping.
Evolving Virtual Creatures The Movie by Karl Sims

Evolved Virtual Creatures

Examples from work in progress
The Golem Project
Automatic Design and Manufacture of Robotic Lifeforms
Hod Lipson and Jordan Pollack

Golem project (Genetically Organized Lifelike Electro Mechanics)

• Simple electro-mechanical systems evolved to yield physical locomoting machines.
• The first time robots have been robotically designed and robotically fabricated.
Golem Project Evolved Creatures

Hod Lipson and Jordan Pollack

Arrow
Ratchet

Tetra
Snake
Golem Project Arrow Creature

Hod Lipson and Jordan Pollack

Simulated Arrow

Real Arrow
Project Objectives

Apply GA to Lego assembly generation.

- Represent Lego assemblies precisely and unambiguously.
- Encode assemblies as a chromosomes.
- Adapt genetic operators.
- Perform Genetic optimization on Lego structures.
Our system was extended from sGA originally created by Prof. Stephen Hartley in 1996 and written on the Java programming language.

Java3D package and VRML97 were used in order to create a visualizer to monitor Lego structures as they evolve.

The system supports:
• One-point crossover
• Proportional, rank, universal stochastic sampling, sigma scaling, and Boltzman selection techniques
• Elitism
• And allows input of the mutation and crossover rates and the population size.
Examples: 10 by 10 by 10 Light Structure

In this case the goal was to evolve a structure with the size of 10 Lego units in each x-y-z dimension with the minimal weight.

- Left structure was created at 895 generation and has sizes 10 by 10 by 6.8
- Right structure was created at 3367 generation and has sizes 10 by 10 by 10.

Both structures among the lightest possible structures that satisfy these parameters that can be created from the set of elements given.
Examples: Pillar-Like Dense Structure

• In this case the goal was to evolve a dense pillar-like structure with the 2 by 4 base and 20, 40 and 60 height.

• Structures shown where evolved in 5000 generations on average.

• All of them exactly match desired size and among densest possible structures.

Height 20, 40 and 60 from right to left
Examples: Staircase Structure

- In this case the goal was to creating staircases of 3, 4, 5, 6, 7 and 10 steps.
- Set of building blocks available reduced to 2 by 1 plates only.
- 7 step staircase was evolved in 51573 generations.
- 10 step staircase was evolved in 568 generations.
- All of them exactly match desired size.

Staircase of 7 steps (top) 10 steps (bottom)
GA Performance Analysis on the Staircase Structure Example

• Total number of solutions 2.
• Total number of Lego structures possible (approx).

\[ 2 \times 7 \times (7 + 3) \times (7 + 3 + 3) \times \ldots \times (7 + 3 \times (n - 1)) = 2 \times \prod_{i=0}^{n-1} (7 + 3 \times i) \]

• Total number of structures examined by GA.

\[ \text{Population Size} \times \text{Number for Generation} \]

<table>
<thead>
<tr>
<th>Number of Steps</th>
<th>Size of Search Space</th>
<th>Number of Solutions</th>
<th>Examined by GA</th>
<th>Examined by GA %</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>29120</td>
<td>2</td>
<td>4100</td>
<td>14%</td>
</tr>
<tr>
<td>5</td>
<td>640640</td>
<td>2</td>
<td>42000</td>
<td>6.5%</td>
</tr>
<tr>
<td>6</td>
<td>16016000</td>
<td>2</td>
<td>127500</td>
<td>0.79%</td>
</tr>
<tr>
<td>7</td>
<td>8520512000</td>
<td>2</td>
<td>5153300</td>
<td>0.06%</td>
</tr>
<tr>
<td>10</td>
<td>332282926976000</td>
<td>2</td>
<td>56800</td>
<td>10^{-7} %</td>
</tr>
</tbody>
</table>